The Disagreement Power of an Adversary

Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Andreas Tielmann

The Disagreement Power of an Adversary

Carole Delporte-Gallet
LIAFA, Université Paris Diderot
Paris, France

Hugues Fauconnier
LIAFA, Université Paris Diderot
Paris, France

Rachid Guerraoui
Distributed Programming Laboratory, EPFL
Lausanne, Switzerland

Andreas Tielmann
LIAFA, Université Paris Diderot
Paris, France

April 20, 2009

Abstract

At the heart of distributed computing lies the fundamental result that the level of agreement that can be obtained in an asynchronous shared memory model where t processes can crash is exactly $t + 1$. In other words, an adversary that can crash any subset of size at most t can prevent the processes from agreeing on t values. But what about the rest $(2^{2^n} - n)$ adversaries that might crash certain combination of processes and not others? Given any adversary, what is its disagreement power? i.e., the biggest k for which it can prevent processes from agreeing on k values.

This paper answers this question. We present a general characterization of adversaries that enables to directly derive their disagreement power. We use our characterization to also close the question of the weakest failure detector for k-set agreement. So far, the result has been obtained for two extreme cases: consensus and $n - 1$-set agreement. We answer this question for any k and any adversary.

1 Introduction

The theory of distributed computing is largely related to determining what can be computed against a specific adversary. Most results so far have been devoted to *one* specific form of adversaries: those that can control any subset of size t of the processes, i.e., the *t-failures* adversary. In particular, a seminal result in distributed computing says that the level of agreement that can be obtained in a shared memory model where t processes can crash is exactly $t+1$ [11, 1, 16]. In other words, an adversary that can crash any subset of size at most t can prevent the processes from agreeing on t values. In the case of consensus for instance ($t=1$), this translates into FLP [8].

In a sense, these results are very incomplete, for the t-failures assumption covers only n cases in a system of size n. What about the rest $(2^{2^n}-n)$ adversaries? i.e., adversaries that can crash certain subsets of processes of a certain size but not others of the same size. Given any adversary $\mathcal{A}$, we seek to compute its power by determining the biggest k for which k-set-agreement [7] cannot be solved, which we call here the *disagreement power* of $\mathcal{A}$. Beyond intellectual curiosity, this question might even be practically motivated by modern multicore architectures where the failures of processes in the same core might all be correlated [14, 9].

The answer is trivial for some adversaries. For others, it is not. Consider, in a system of 3 processes, $\{1,2,3\}$, an adversary $\mathcal{A}$ that can fail either no process, both processes 2 and 3, or process 1. It is easy to show that $\mathcal{A}$ can prevent consensus but not 2-set agreement. In this sense, adversary $\mathcal{A}$ has the same disagreement power as the 1-failure adversary, i.e., 1. Consider now a more involved scenario: a system of 4 processes and an adversary $\mathcal{A}$' that can fail any element of $\{\emptyset, 4, 23, 14, 12, 134, 124, 123\}$. What is the disagreement power of $\mathcal{A}$'? We prove in this paper that it is also 1.

The first contribution of this work is a general characterization of adversaries that enables to directly derive their disagreement power. Namely, we introduce a *structure predicate*, parameterized by an integer k, and which, intuitively, ensures that for any set of faulty processes of size less or equal k, there is some adequate *matching* set of those that can be crashed by the adversary. We exploit this through a new simulation of adversaries, which we call the *conservative back-off simulation*. This basically shows that if k-set agreement can be solved with some adversary that satisfies the predicate for some k, then k-set agreement can be solved with the k-failure adversary. Hence, an adversary that satisfies the predicate has disagreement power at least k. In our simulation, a process backs-off and skips its simulation step if it thinks that it is faulty in some set where the simulated algorithm is known to work. Conversely, we show how to solve k-set agreement with any adversary $\mathcal{A}$ that does not satify the predicate for some k. We do this by showing how to implement failure detector k-anti-Ω [17], known in turn to implement k-set-agreement. (Each query to k-anti-Ω returns $n-k$ process ids; the specification ensures that there is a correct process whose id is eventually never output.)

Our characterization hence splits the set of all adversaries into n disjoint (*equivalence*) classes, one for every level of disagreement. It contributes to the idea that a very small subset of results and ad-hoc proofs in distributed computing should suffice to derive all others. In particular, if indeed needed to reason about set-agreement for the "wait-free" adversary ($n-1$-set agreement), topology is not needed for all the other ones. Results concerning other k-fault adversaries can be deduced by [3, 4], whereas results for all other $(2^{2^n}-n)$ adversaries can be deduced from our characterization.

The second contribution of our paper, also obtained from our characterization, is to close the question of the weakest failure detector for k-set agreement. So far, the result has been obtained for two extreme cases: consensus [5, 15] and $n-1$-set agreement [10, 17]. We close this long lasting question for *any* k and *any* adversary. More specifically, we show that k-anti-Ω is the

weakest failure detector to solve a decision problem that is impossible with the k-adversary. In this sense, k-set agreement is the weakest impossible decision problem for any adversary that has disagreement power k. We give a general proof that reuses some of the ideas of [5, 10, 17] together with a fundamental observation: the fact that a breath search extraction technique is enough to extract the weakest failure detector because, even if we can extract only a specific adversary, we can compute its disagreement power using our characterization technique. In short, we proceed as follows: (1) we locally simulate a graph of block runs of some shape; (2) we conduct a breath search and extract a set of adversaries that contain the actual set of faulty processes; (3) we show that such an adversary cannot satisfy our structure predicate; (4) finally we extract k-anti-Ω from it.

The rest of the paper is structured as follows. We first define our structural predicate in Section 2. We then introduce our conservative back-off simulation and use it in Section 3.1 to show that any adversary that satisfies the predicate for k can be reduced to the k-failure adversary. We then show in Section 3.2 how to implement k-set agreement with any adversary that does not satisfy the predicate, In Section 4, we determine the weakest failure detector for k-set agreement for any adversary $\mathcal{A}$.

2 Model and Definitions

We consider the standard shared-memory model [12, 13]. In Section 4, this model is augmented with failure detectors [6]. We recall below the necessary elements to describe our and introduce the notion of adversary.

Processes and objects. The system consists of n processes $\Pi = \{p_1, p_2, ..., p_n\}$ which communicate using read-write atomic registers. Processes might crash. In this case we say that the process is faulty. A process that never crashes is said to be correct.

Failure pattern and adversary. We assume the existence of a global discrete clock T that is, however, inaccessible to the processes. A failure pattern F [6] is a function from a global time range T to 2^{Π}, where $F(t)$ denotes the set of processes that have crashed by time t. The set of correct processes is $Correct(F) = \Pi \setminus \cup_{t \in T} F(t)$.

Intuitively, an *adversary* chooses which sets of processes may crash. More precisely, an adversary can be represented by a set of sets of processes (we call these sets *faulty-sets*). Given an adversary $\mathcal{A}$, $FP(\mathcal{A})$, the set of failure patterns associated with $\mathcal{A}$, is the set of all failure patterns F such that $\Pi \setminus Correct(F)$ is one of the faulty-sets of $\mathcal{A}$. Here we consider only adversaries $\mathcal{A}$ for which there is always at least one correct process, namely such that $\Pi \notin \mathcal{A}$. The k-adversary, denoted $\mathcal{B}_k$, is the adversary for which at most k $(0 \leq k \leq n-1)$ processes may crash: $\mathcal{B}_k = \{b \subseteq \Pi \mid |b| \leq k\}$. For a set S of sets of processes, we define adversary $\mathcal{A}(S)$ to be the set $\{a \subseteq \Pi \mid \exists B \in S, a = \Pi \setminus B\}$.

To define how we compare adversaries, we first discuss how their faulty-sets are compared:

Definition 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be any two adversaries. We say that a faulty-set $a \in \mathcal{A}$ dominates a faulty-set $b \in \mathcal{B}$ for $\mathcal{A}$ and $\mathcal{B}$, if and only if*

$$\begin{aligned} & a \supseteq b \\ \textit{and} \quad & \forall b' \in \mathcal{B}, b' \supsetneq b, \exists a' \in \mathcal{A}, a' \supseteq a, a' \textit{ dominates } b' \textit{ for } \mathcal{A} \textit{ and } \mathcal{B} \end{aligned}$$

Definition 2. *Adversary $\mathcal{A}$ is stronger than adversary $\mathcal{B}$, if and only if the following property $\boldsymbol{P}$ is satisfied:*

$$\boldsymbol{P}(\mathcal{A}, \mathcal{B}): \quad \forall b \in \mathcal{B}, \exists a \in \mathcal{A} \textit{ and } a \textit{ dominates } b \textit{ for } \mathcal{A} \textit{ and } \mathcal{B}$$

When we compare an adversary to the k-adversary $\mathcal{B}_k$, we simply write $\mathbf{P}_k(\mathcal{A})$ instead of $\mathbf{P}(\mathcal{A}, \mathcal{B}_k)$.

Algorithms. An algorithm A consists of a set of n deterministic automata, $(A(p_1), \ldots, A(p_n))$ one for each process p_i in the system. Computation proceeds in atomic steps and each atomic step is a step of one of the $A(p_i)$'s. A process executes the algorithm assigned to it, until the process crashes and stops executing any action.

Configuration and runs. A configuration of algorithm A defines the state of each process and each shared register in the system. In an initial configuration I of A, every process p_i is in an initial state of $A(p_i)$ and every register is in an initial state. A run of algorithm A with adversary $\mathcal{A}$ is a tuple $R = (F, I, S, T)$ where F is a failure pattern belonging to $FP(\mathcal{A})$, I is an initial configuration of A, S is an infinite sequence of steps of A, T is an infinite list of non-decreasing time values indicating when each step of S has occurred such that if $T(t)$ is a step of p_i then p_i does not belong to $F(t)$ and every p_i belonging to $Correct(F)$ makes an infinity of steps in S.

Problems and k-set agreement. Generally, we say that algorithm A solves problem P for adversary $\mathcal{A}$ if and only if every run $R = (F, I, S, T)$ of A with $F \in FP(\mathcal{A})$ satisfies the specification of P (we say also A implements P for adversary $\mathcal{A}$).

In the following we restrict ourselves to specific decision problems for which the termination condition depends only on global configurations (the states of each process and shared memory) of the system. These clearly include k-set agreement. We give a specification for the k-set agreement task specification. k-set agreement is a decision task in which processes decide by writing their decision value in shared registers (one register by process) that satisfies the following properties: 1.*Agreement*: At most k different values are decided; 2. *Termination*: At least one process eventually decides ; 3. *Validity*: If any process decides a value v, then v is the initial value of some process.

It is easy to verify that the termination condition depends only on the global configuration of the system and that the given specification is equivalent to the classical one [7] in which all correct processes have to decide.

Recall that k-set agreement can be solved in $\mathcal{B}_x$ if $0 \leq x \leq k-1$. But not in $\mathcal{B}_x$ if $k \leq x \leq n-1$ [11, 1, 16].

Failure detectors. In addition to performing read and write operations, a process can query a failure detector object [6]. We say that algorithm A with failure detector $\mathcal{D}$ solves a problem P for adversary $\mathcal{A}$, if every run $R = (F, H, I, S, T)$ as classically defined of A with $F \in FP(\mathcal{A})$ satisfies the specification of P (when the context is clear we also say $\mathcal{D}$ implements P).

Note that from an algorithm with failure detector $\mathcal{D}$ that solves problem P for adversary $\mathcal{A}$ and an algorithm that implements failure detector $\mathcal{D}$ for adversary $\mathcal{A}$, we obtain an algorithm that solves the problem for adversary $\mathcal{A}$.

Recall that failure detectors can be partially ordered: failure detector $\mathcal{D}$ is said to be stronger than failure detector $\mathcal{D}'$ if there exists an algorithm with $\mathcal{D}$ that implements $\mathcal{D}'$. Failure detector $\mathcal{D}$ is said to be weakest to solve a problem P if (1) $\mathcal{D}$ implements P and, (2) $\mathcal{D}$ is stronger that any failure detector $\mathcal{D}'$ that implements P.

In this paper, we consider failure detector k-anti-Ω [17]: each query to k-anti-Ω returns $n-k$ process ids, with the guarantee that there is a correct process whose id is returned only a finite number of times. If $k = 1$, k-anti-Ω is equivalent to the eventual leader Ω failure detector, the weakest failure detector for consensus [5, 15]. If $k = n-1$, k-anti-Ω is anti-Ω, the weakest failure detector to solve the $n-1$-set agreement [17]. Zielinsky in [17] conjectured that k-anti-Ω is the

weakest failure detector to solve k -set agreement. Our paper proves this conjecture in Section 4.

3 Disagreement Power of an Adversary

3.1 The Conservative Back-off Simulation

Assume property $\mathbf{P}(\mathcal{A}, \mathcal{B})$ is fulfilled for some adversaries $\mathcal{A}$ and $\mathcal{B}$. Let Alg be any algorithm which solves a decision problem P in $\mathcal{A}$. Then, the *conservative back-off simulation* with Alg in Figure 1 solves P in $\mathcal{B}$.

The goal of the simulation is to identify, in every possible run with a set of faulty processes $b \in \mathcal{B}$, a set of faulty processes $a \in \mathcal{A}$ with $b \subseteq a$ (i.e. more failures in a than in b). Hence, the processes outside a can use the given algorithm which is known to work for every $a \in \mathcal{A}$. The processes in a that are outside b can then just back-off and omit to take simulation steps, since the others are enough to ensure termination.

To determine a, we first estimate possible b's in the run. This is achieved by simply using step-counters. The current estimations are stored in variables $b_1, \ldots b_l$. Then, beginning with the smallest set b_1, every process tries to stepwise narrow down the possibilities for a. In these steps, our structural predicate $\mathbf{P}(\mathcal{A}, \mathcal{B})$ is needed. It is guaranteed that there is always an $a \in \mathcal{A}$ that is a superset of all a's considered so far. This sequence of a's is stored in *faulty*. Therefore, even if the estimations of b oscillate, the minimal a is such that $b \subseteq a$ is stable (with b beeing the actual set of faulty processes). Therefore, it is safe for the processes to take steps if they do not belong to some $a \in$ *faulty* and have reason to believe that all other processes that are not in a are alive.

---

initially:

1 $Stepc_i := 0$; (∗ a SWMR register ∗)
2 $faulty_i := \emptyset$; (∗ estimates for faulty ∗)
3 $alive_i := \emptyset$; (∗ estimate of non-faulty ∗)
4 $lastsimsteps_i := [0, \ldots, 0]$; (∗ the state at the last simulated step ∗)

forever do:

5 let $p_{i_1}, \ldots, p_{i_n}$ be the processes ordered by increasing *Stepc* (ties broken deterministically);
6 $l := 1$; $faulty_i := \emptyset$;
7 **if** $\emptyset \in \mathcal{B}$ **then** $b_1 := \emptyset$; inc(l);
8 **for** $j = 1, \ldots, n$ **do if** $\{p_{i_1}, \ldots, p_{i_j}\} \in \mathcal{B}$ **then** $b_l := \{p_{i_1}, \ldots, p_{i_j}\}$; inc($l$);
9 **for** $j = 1, \ldots, l$ **do**
10 add smallest $a \in \mathcal{A}$ to $faulty_i$ s.t. a domin. b_j and $\forall a' \in faulty_i, a \supseteq a'$;
11 $alive_i := \{p_j \mid Stepc_j > lastsimsteps_i[j]\}$;
12 **if** $\exists a \in faulty_i, alive_i \cup a = \Pi$ and $p_i \notin a$ **then**
13 execute a step of Alg;
14 $lastsimsteps_i := [Stepc_1, \ldots, Stepc_n]$;
15 $Stepc_i := Stepc_i + 1$;

Figure 1: The conservative back-off simulation for process p_i and $P(\mathcal{A}, \mathcal{B})$.

Theorem 1. *If $\mathbf{P}(\mathcal{A}, \mathcal{B})$, every decision problem which can be solved with $\mathcal{A}$ can also be solved with $\mathcal{B}$.*

Proof. We show that the simulation algorithm in Figure 1 decides for any algorithm Alg and any problem P in all runs in $\mathcal{B}$, i.e. that the simulation of Alg decides.

Let $b^* \in \mathcal{B}$ be the actual set of faulty processes in some run. Then, there exists some $j^* > 0$ such that b^* is eventually always b_{j^*} at all correct processes, because all the step counters at processes in sets b_j with $j \leq j^*$ change only finitely often and the step counters of some of the processes in all other sets increase infinitely often. Furthermore, all b_j with $j < j^*$ are eventually also always the same at all processes.

Let $a^* \in \mathcal{A}$ be the smallest set with $a^* \supseteq b^*$ that is eventually always added to $faulty_i$. By property $P(\mathcal{A}, \mathcal{B})$ such a set has to exist (e.g. the one that is added in step j^*).

Then, eventually, and at all correct processes, for all sets $a \in faulty_i$ where $a \cup alive_i = \Pi$, a is a superset of a^*, because for all strict subsets a' there is at least one process $p \notin a'$ that makes only finitely many steps. Since eventually only processes that are not in such an a take steps, processes in a^* simulate only finitely many steps of Alg.

Assume some process p_j that is not in a^* simulates only finitely many steps of Alg. Since $a^* \supseteq b^*$, all these processes take infinitely many steps. Therefore, eventually, $alive_j \supseteq \Pi \setminus a^*$. But then, $alive_j \cup a^* = \Pi$. A contradiction to the fact that p_j simulates only finitely many steps of Alg.

Therefore, exactly the processes not in a^* simulate infinitely many steps. Since $a^* \in \mathcal{A}$, Alg has to terminate. □

3.2 k-Set Agreement

In this Section, we compare an adversary $\mathcal{A}$ with the adversary that contains all sets of size less or equal k. We show that if $\mathbf{P}_k(\mathcal{A})$ does not hold for some adversary $\mathcal{A}$ then it is possible to implement k-set agreement for $\mathcal{A}$. For this, it is sufficient to show how to implement k-anti-Ω, since this is sufficient to implement k-set agreement for any adversary [17]. Basically, k-anti-Ω outputs, whenever queried, at least $n - k$ processes, s.t. at least one correct process is output only finitely often. The algorithm in Figure 2 implements k-anti-Ω.

As in the previous section, we first estimate possible sets of faulty processes in the run. This is achieved by using step-counters. The current estimations are stored in variables $a_1, \ldots a_l$. Then, we take the smallest set $b_{init} \in \mathcal{B}_k$ that is not dominated by any $a \in \mathcal{A}$ (by predicate $\neg\mathbf{P}_k(\mathcal{A})$, there exists one). Although this set is not dominated by any a, it may contain no correct process (in particular, b_{init} may be the empty set). However, if so, then by the recursive nature of the domination property, there has to exist a strict superset of b_{init} which is not dominated by all $a \in \mathcal{A}$ with $a \supseteq b_{init}$ (if $b_{init} = \emptyset$, then this applies to all $a \in \mathcal{A}$). By an iterated use of this property for every possible a_j, eventually we end up with an element of $b \in \mathcal{B}_k$ which contains a correct process (because $\neg(a \supseteq b)$). Although this output may differ in each round, it will eventually always contain at least one common correct process, because some prefix of $a_1, \ldots a_l$ is stable.

Theorem 2. *If $\boldsymbol{P}_k(\mathcal{A})$ does not hold for $\mathcal{A}$, then it is possible to implement k-anti-Ω in $\mathcal{A}$.*

Proof. If $\mathbf{P}_k(\mathcal{A})$ does not hold, then

$$\exists b \in \mathcal{B}_k, \forall a \in \mathcal{A},\ a \text{ does not dominate } b \text{ for } \mathcal{A} \text{ and } \mathcal{B}_k$$

and if a does not dominate b for $\mathcal{A}$ and $\mathcal{B}_k$, then

$$\begin{aligned} & \neg(a \supseteq b) \\ \vee\ & \exists b' \in \mathcal{B}, b' \supsetneq b, \forall a' \in \mathcal{A}, a' \supseteq a, a' \text{ doesn't dominate } b' \text{ for } \mathcal{A} \text{ and } \mathcal{B} \end{aligned}$$

Let $a^* \in \mathcal{A}$ be the actual set of faulty processes in some run. There exists some $j^* > 0$ such that a^* is eventually always a_{j^*} at all processes, because all the step counters at processes in sets a_j with $j \leq j^*$ change only finitely often and the step counters of some of the processes in all other sets increase infinitely often.

initially:

1 $Stepc_i := 0$; (∗ a SWMR register ∗)
2 $S_i := \emptyset$;

forever do:

3 let $p_{i_1}, \ldots, p_{i_n}$ be the processes ordered by increasing *Stepc* (ties broken deterministically);
4 $l := 1$; **if** $\emptyset \in \mathcal{A}$ **then** $a_1 := \emptyset$; inc(l);
5 **for** $j = 1, \ldots, n$ **do if** $\{p_{i_1}, \ldots, p_{i_j}\} \in \mathcal{A}$ **then** $a_l := \{p_{i_1}, \ldots, p_{i_j}\}$; inc($l$);
6 $S_i :=$ the smallest $b_{init} \in \mathcal{B}_k$ s.t. no $a \in \mathcal{A}$ does dominate it;
7 **for** $j = 1, \ldots, l$ **do**
8 **while** $a_j \supseteq S_i$ **do**
9 $S_i :=$ the smallest $b \in \mathcal{B}_k, b \supsetneq S_i$ s.t. $\forall a \supseteq a_j : \neg(a$ dom. $b)$;
10 **if** $|S_i| < k$ **then** add some processes to S_i until $|S_i| = k$;
11 output $\Pi \setminus S_i$;
12 $Stepc_i := Stepc_i + 1$;

Figure 2: Implementation of k-anti-Ω

By property $\neg\mathbf{P}_k(\mathcal{A})$, there exists some good b_{init}. Let $b^* \supseteq b_{init}$ be the maximal set such that b^* is eventually always a subset of S_i at the end of the for-loop and assume $a^* \supseteq b^*$. Then, in step j^* of the for-loop, there is no $b \supsetneq b^*$ with which $S_i = b^*$ can be replaced. Therefore, $\forall b \in \mathcal{B}_k, b \supsetneq b^*, \exists a \supseteq a_{j^*}$, a dom. b. Thus, a_{j^*} dom. b^* and we have a contradiction to the conditions under which a set is added to S_i or to the initial condition of b_{init}.

Therefore, $\neg(a^* \supseteq b^*)$ and there exists a process $p \in b^*$ which is not in a^* and the properties of k-anti-Ω are fulfilled. □

4 Weakest Failure Detector

4.1 Preliminaries

We present our results in the same framework as Zielinski [17]. We only recall here the main points. We consider the immediate atomic snapshot model [2], which is equivalent to the shared memory model, but allows a higher level of abstraction. Specifically, in this model a run can be represented as a *block run*: a sequence of sets (called blocks) in which the snapshot operations of the processes in one block appear to take take effect at the same time.

We use several building blocks, developed in [17]. In particular, we consider means to collect failure detector values in a causal precedence relation in order to simulate steps of processes in a block run such that a run composed by such steps is indistinguishable from some real run.

An immediate atomic snapshot register is divided into n fields, one for each process. The process computation is specified by a function *localcomp*. This function gets passed the current state of all fields in the immediate atomic snapshot register, together with an optional failure detector value. A process can perform its computation using these values and then return a value which it wants to be written into its corresponding field in the register. Then, for some block B in a block run, every process $p \in B$ sees the same state of the register where all operations of all previous blocks have taken place and the value returned by the last invocation of *localcomp* at p is written to the register.

Note that if the parameters of *localcomp* contain also failure detector outputs, a run is not

anymore solely determined by the block run, it furthermore depends also on the failure detector history.

A process cannot determine exactly in which blocks it appears in some run. Nevertheless, by using a simple step-counter, a process can detect which processes took steps between two steps of the same process. But it cannot immediately determine if the steps of these processes belong to the same block or to some sequence of blocks. To reflect this uncertainty, we – in the following definition – do not distinguish between sequences of blocks of the same processes where no process appears twice.

Definition 3. *A block sequence* $B_1, B_2, \ldots$ *is* built upon *a set of blocks* S*, iff for every* B_i*,* $i \geq 1$ *there exists a* $B \in S$ *such that:*

$$B_i = B \tag{1}$$

$$\vee \quad \exists l_1, l_2,\ l_1, l_2 > 0 : (\cup_{1 \leq i-l_1 \leq i \leq i+l_2} B_i = B \wedge \cap_{1 \leq i-l_1 \leq i \leq i+l_2} B_i = \emptyset) \tag{2}$$

4.2 Outline

Let $\mathcal{A}$ be some adversary with $\mathbf{P}_k(\mathcal{A})$. Since k-anti-Ω is impossible to implement in $\mathcal{B}_k$, it is also impossible in $\mathcal{A}$. To extract the weakest failure detector for any decision problem that cannot be solved for adversaries with disagreement power k, we assume that we have given an algorithm $Algorithm_\Delta$ with some failure detector Δ that solves such a decision problem in $\mathcal{A}$ that is impossible without Δ.

We start by collecting failure detector samples from the current run and storing their causal relations in some graph (exactly as in [17]). With this graph, steps of $Algorithm_\Delta$ can be emulated. To get approximations of the failure pattern of the current run, we seek to simulate all possible runs for all possible adversaries in Figure 3 to see which runs terminate.

By using a similar approach as in Figure 1, we show that if the approximations of the extracted adversaries do not have certain good properties, this would contradict the impossibility of the problem in $\mathcal{B}_k$ (which follows from the impossibility of the problem in $\mathcal{A}$). Therefore, we exploit these properties, using a similar approach as in Figure 2 to implement k-anti-Ω. Thereby showing that k-anti-Ω is the weakest failure detector to solve the problem with $\mathcal{A}$.

4.3 Construction

All runs in which only correct processes take infinitely many steps have to terminate. However, enumerating all these runs may never terminate, since there may be no bound on the number of steps taken in these runs. To circumvent this problem, we simulate only runs where every block B corresponds to some set a in the currently investigated adversary (i.e. $B = \Pi \setminus a$). With this approach, for adversary $\mathcal{A}_c = \{\{\Pi \setminus \mathit{correct}\}\}$, only finitely many runs have to be generated (Lemma 4). The approach does allow to derive conclusions about an adversary only for a subset of all possible runs. For example, processes that are not correct can take finitely many steps in some run and even after all faulty processes have crashed, the blocks in the run may be composed of subsets of the set of correct processes, as long as every correct process appears infinitely often in some of these subsets.

Fortunately, the latter problem can be dealt with by exploiting some knowledge about the adversary to force processes to "wait" for appropriate blocks (this is done in Figure 5). To cover also the steps of faulty processes in some run, the block runs for some adversary are not only explored for one initial state, but for a permanently updated set I of initial states. However, this

initially:

1 s_0 := the initial state of Algorithm$_\Delta$;
2 $\forall b \in \mathcal{B}_k, blockc[b] := 0$;
3 $I := \{s_0\}$;
4 $Sets := \{S \subseteq 2^\Pi \mid (S \neq \emptyset) \wedge (\emptyset \notin S) \wedge (\forall B, B' \in S, B \cup B' \in S)\}$;
5 $\forall S \in Sets, suspc[S] := 0$;
6 $\forall S \subseteq 2^\Pi, I_S := I$;

function explore(S, s_j):

7 $visited := \emptyset$;
8 **if** no decision in state s_j **then**
9 for all $B \in S$ do
10 for every permutation of every partitioning $B_1, \ldots, B_m$ of B do
11 simulate the steps for $B_1, \ldots, B_m$ from state s_j and record the new state as s_{j+1} (**if** some step aborts, **then** *aborted* := true; return $\{s_j\}$);
12 $visited := \{s_i\} \cup$ return value of explore(S, s_{j+1});
13 return *visited*;

forever do:

14 permanently keep updating the causal precedence relation of the failure detector values;
15 *changed* := processes which made a step since the last round;
16 $I :=$ explore$(2^\Pi \setminus \{\emptyset\}, s_o)$;
17 *suspected* := $\emptyset$;
18 for all $S \in Sets$ do:
19 *aborted* := false;
20 **if** $\exists p \in changed, \exists B \in S, p \notin B$ **then** $I_S := I$;
21 for all $s_o \in I_S$ do: explore(S, s_o);
22 **if** *aborted* = true **then** add S to *suspected*;
23 $\forall S \in suspected$: inc$(suspc(S))$;
24 $C = \{S \mid S \notin suspected\}$;
25 $\forall b \in \mathcal{B}_k$: **if** $\forall S \in C, \forall a \in \mathcal{A}(S), \neg(a$ dom. $b)$ in $\mathcal{A}(S), \mathcal{B}_k$ **then** inc$(blockc[b])$;
26 k-anti-Ω output := output of alg. in Figure 4 with C and b_{init} as the $b \in \mathcal{B}_k$ with the maximal block-counter;

Figure 3: Extraction of the sub-adversaries and emulation of k-anti-Ω.

again may cause problems for adversary $\mathcal{A}_c$. The tree of all runs for one initial state is finite for $\mathcal{A}_c$, but the set of initial states may grow faster than all these trees can be explored. Therefore, we take advantage of one difference between $\mathcal{A}_c$ and most other adversaries. We update the set I of initial states only for adversaries, if we observe that some process which is faulty in one set of such an adversary has made a step. In this way, the set of initial states for $\mathcal{A}_c$ is updated only finitely often. Again, this does not come for free. The set of initial states of some adversaries which contain only supersets of the set of correct processes are also updated only finitely often. However, fortunately these adversaries do not impact the correctness of the algorithm in Figure 4 (because they do not impact the "good property" we spoke about above).

This property is that we have some set of possible adversaries where the algorithm terminates. If there is one set of the k-failure adversary $\mathcal{B}_k$ which is not dominated by any set of some extracted adversary, then, using a similar approach as in Figure 2, we can extract k-anti-Ω with the algorithm in Figure 4.

To prove that this property indeed holds, we reduce it to an impossible problem. If every set of

the k-failure adversary $\mathcal{B}_k$ is dominated by some set of some extracted adversary, then, by using a similar approach as in Figure 1, in Figure 5 an algorithm that solves the impossible problem with $\mathcal{B}_k$ is constructed. Therefore, Figure 4 implements k-anti-Ω.

4.4 Proof

Let $\mathcal{F}$ be the failure pattern of some run and $O = \{S_1, \dots S_m\}$ be the set of all sets that are infinitely often not suspected in the run and let O' be the maximal subset of O such that for all $S \in O', \forall B \in S, B \subseteq \mathit{correct}(\mathcal{F})$.

Lemma 1. $\forall S \in O \setminus O', \forall B \in S, \mathit{correct}(\mathcal{F}) \subseteq B$.

Proof. Assume $S \in O \setminus O'$ and assume there exists a set $B' \in S$ that is not a superset of $\mathit{correct}(\mathcal{F})$. Thus, I_S gets infinitely often updated, because there is some process p, p not in B' that takes infinitely many steps.

Since $S \in O \setminus O'$, there exists some $B \in S$ that contains a faulty process q. Eventually, the simulation runs out of failure detector samples for q, because p generates infinitely many initial states in I_S. A contradiction to $S \in O$. □

Lemma 2. *Every run in which the set of processes that take infinitely many steps is* $\mathit{correct}(\mathcal{F})$*, terminates.*

Proof. Follows directly by the assumption about $\mathit{Algorithm}_{\Delta}$. □

Lemma 3. *Every run in which only processes in* $\mathit{correct}(\mathcal{F})$ *take steps, which is eventually built upon some* $S \in O'$ *terminates.*

Proof. If $S = \{\mathit{correct}(\mathcal{F})\}$, then this follows directly from Lemma 2. Otherwise, I_S is updated infinitely often, because S contains a set in which some correct process is missing. Therefore, every prefix in which only processes in $\mathit{correct}(\mathcal{F})$ take steps is eventually generated.

Assume for one such a prefix that there exists a non-terminating run which is eventually built upon some $S \in O'$. Then, "explore" will eventually always abort for set S. A contradiction to $S \in O'$. □

Lemma 4. *The set* $\{\mathit{correct}(\mathcal{F})\}$ *is only finitely often suspected.*

Proof. Since processes that are not in $\mathit{correct}(\mathcal{F})$ take only finitely many steps, $I_{\{\mathit{correct}(\mathcal{F})\}}$ is only finitely often updated. Consider the tree of all non-terminating runs that start from one state in $I_{\{\mathit{correct}(\mathcal{F})\}}$ and are built upon $\{\mathit{correct}(\mathcal{F})\}$. In this tree, all nodes have a finite degree.

Assume by contradiction that this tree has infinitely many nodes. Then, by Königs Lemma, the tree contains an infinite path. This means there exists an infinite path in which the set of all processes that take infinitely many steps is exactly $\mathit{correct}(\mathcal{F})$. A contradiction to Lemma 2. Therefore, the tree has only finitely many nodes and "explore" eventually does not abort for $S = \{\mathit{correct}(\mathcal{F})\}$. □

For all $S \in O'$, let $\mathcal{A}(S)$ be $\{a \subseteq \Pi \mid \exists B \in S, a = \Pi \setminus b\}$ and $\mathcal{A}(O') := \cup_{S \in O'} \mathcal{A}(S)$.

Theorem 3. *Let* c *be a set of processes and assume that*

1. *for all* $b \in \mathcal{B}_k, b \neq \emptyset, \exists S \in O', \exists B \in S :$ $(\Pi \setminus B)$ *dom. b in* $\mathcal{A}(S), \mathcal{B}_k$.
2. *every run in which only processes in* c *take steps, which is eventually built upon some* $S \in O'$ *terminates.*

initially:

1 $Stepc_i := 0$; (∗ a SWMR register ∗)
2 $S_i := \emptyset$;

forever do:

3 let $p_{i_1}, \ldots, p_{i_n}$ be the processes ordered by increasing *Stepc* (ties broken deterministically);
4 $l := 1$; $S_i := b_{init}$; **if** $\emptyset \in \mathcal{A}$ **then** $a_1 := \emptyset$; inc(l);
5 **for** $j = 1, \ldots, n$ **do if** $\{p_{i_1}, \ldots, p_{i_j}\} \in \mathcal{A}$ **then** $a_l := \{p_{i_1}, \ldots, p_{i_j}\}$; inc($l$);
6 **for** $j = 1, \ldots, l$ **do**
7 **while** $a_j \supseteq S_i$ **do**
8 $S_i :=$ some $b \in \mathcal{B}_k, b \supsetneq S_i$ s.t. for the S with $a_j \in \mathcal{A}(S)$ with minimal $suspc[S]$, $\forall a \in S$ with $a \supseteq a_j$, $\neg(a$ dom. $b)$
9 **if** $|S_i| < k$ **then** add some processes to S_i until $|S_i| = k$;
10 output $\Pi \setminus S_i$;
11 $Stepc_i := Stepc_i + 1$;

Figure 4: Implementation of k-anti-Ω for a non-stable set C

3. every run in which the set of processes that take infinitely many steps is c terminates.

Then, the algorithm in Figure 5 terminates in all runs of adversary $\mathcal{B}_k$.

Proof. The proof is analogous to the proof of Theorem 1. If no process is faulty in some run, then by property 3, the algorithm terminates. If some process is faulty, then by property 1, S is eventually stable and by property 2, the algorithm terminates also. □

Lemma 5. *If there exists some $b \in \mathcal{B}_k, b \neq \emptyset, \forall S \in O', \forall B \in S : (\Pi \setminus B)$ does not dom. b in $\mathcal{A}(S), \mathcal{B}_k$, then the algorithm in Figure 4 emulates k-anti-Ω.*

Proof. The proof is analogous to the proof of Theorem 2. Since there exists a $b \in \mathcal{B}_k$, $b \neq \emptyset$, $\forall S \in O'$, $\forall B \in S$: $(\Pi \setminus B)$ does not dom. b in $\mathcal{A}(S), \mathcal{B}_k$, this b is also not dominated by the sets in $O \setminus O'$, because it is not by $\{correct(\mathcal{F})\}$. Furthermore, for all supersets of $correct(\mathcal{F})$, the order in *suspc* is stable. Therefore, for $a_j = \Pi \setminus correct(\mathcal{F})$, b is still available and eventually always the same b is selected.

□

Theorem 4. *The algorithm in Figure 3 emulates k-anti-Ω.*

Proof. If there exists some $b \in \mathcal{B}_k$, $b \neq \emptyset$, $\forall S \in O'$, $\forall B \in S$: $(\Pi \setminus B)$does not dom. b in $\mathcal{A}(S), \mathcal{B}_k$, then the algorithm in Figure 4 emulates k-anti-Ω (Lemma 5).

Assume for every $b \in \mathcal{B}_k$ with $b \neq \emptyset$ there exists some $S \in O'$, and some $B \in S$ such that $(\Pi \setminus B)$ dom. b in $\mathcal{A}(S), \mathcal{B}_k$. Then, with Lemma 3 and 2 and $c = correct(\mathcal{F})$, the preconditions for Theorem 3 are fulfilled. But this contradicts the impossibility of the problem in $\mathcal{B}_k$. Therefore, k-anti-Ω is correctly emulated. □

References

[1] Elizabeth Borowsky and Eli Gafni. Generalized flp impossibility result for t-resilient asynchronous computations. In *STOC*, pages 91–100, 1993.

initially:

1 $Stepc_i := 0$; (∗ a SWMR register ∗)
2 $faulty_i := \emptyset$; (∗ estimates for faulty ∗)
3 $alive_i := \emptyset$; (∗ estimate of non-faulty ∗)
4 $lastsimsteps_i := [0, \ldots, 0]$; (∗ the state at the last simulated step ∗)

forever do:

5 let $p_{i_1}, \ldots, p_{i_n}$ be the processes ordered by increasing $Stepc$ (ties broken deterministically);
6 $b_1 := \emptyset$;
7 **for** $j = 2, \ldots, n$ **do if** $\{p_{i_1}, \ldots, p_{i_j}\} \in \mathcal{B}_k$ **then** $b_l := \{p_{i_1}, \ldots, p_{i_j}\}$; inc($l$);
8 $faulty_i := \{\Pi \setminus c\}$; (∗ $\Pi \setminus c$ dom. $(b_1 = \emptyset)$) ∗)
9 let $S \in O'$ be minimal such that $\exists B \in S$, $(\Pi \setminus B)$ dom. b_2 in $\mathcal{A}(S), \mathcal{B}_k$;
10 **for** $j = 3, \ldots, l$ **do**
11 add smallest $a \in \mathcal{A}(S)$ to $faulty_i$ s.t. a domin. b_j in $\mathcal{A}(S), \mathcal{B}_k$ and $\forall a' \in faulty_i, a \supseteq a'$;
12 $alive_i := \{p_j \mid Stepc_j > lastsimsteps_i[j]\}$;
13 **if** $\exists a \in faulty_i$, $alive_i \cup a = \Pi$ and $p_i \notin a$ **then**
14 execute a step of $Algorithm_\Delta$ with the sampling of the simulations (skip the step, if there is some process $q \neq p_i$, $q \notin a$ that has made a simulated step since the last simulation step of p_i);
15 $lastsimsteps_i := [Stepc_1, \ldots, Stepc_n]$;
16 $Stepc_i := Stepc_i + 1$;

Figure 5: The extended back-off simulation for process p_i and $P(\mathcal{A}(O'), \mathcal{B}_k)$.

[2] Elizabeth Borowsky and Eli Gafni. Immediate atomic snapshots and fast renaming. In *PODC '93: Proceedings of the twelfth annual ACM symposium on Principles of distributed computing*, pages 41–51, New York, NY, USA, 1993. ACM.

[3] Elizabeth Borowsky, Eli Gafni, Nancy A. Lynch, and Sergio Rajsbaum. The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127–146, 2001.

[4] Tushar Deepak Chandra, Vassos Hadzilacos, Prasad Jayanti, and Sam Toueg. Generalized irreducibility of consensus and the equivalence of t-resilient and wait-free implementations of consensus. *SIAM J. Comput.*, 34(2):333–357, 2004.

[5] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. In Maurice Herlihy, editor, *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing (PODC'92)*, pages 147–158, Vancouver, BC, Canada, 1992. ACM Press.

[6] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[7] Soma Chaudhuri. Agreement is harder than consensus: set consensus problems in totally asynchronous systems. In *Proceedings of Principles of Distributed Computing 1990*, 1990.

[8] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[9] Matthias Fitzi and Ueli M. Maurer. Efficient byzantine agreement secure against general adversaries. In Shay Kutten, editor, *DISC*, volume 1499 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 1998.

[10] Rachid Guerraoui, Maurice Herlihy, Petr Kouznetsov, Nancy A. Lynch, and Calvin C. Newport. On the weakest failure detector ever. In Indranil Gupta and Roger Wattenhofer, editors, *PODC*, pages 235–243. ACM, 2007.

[11] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.

[12] Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

[13] Prasad Jayanti. Robust wait-free hierarchies. *J. ACM*, 44(4):592–614, 1997.

[14] Flavio Paiva Junqueira and Keith Marzullo. Designing algorithms for dependent process failures. In André Schiper, Alexander A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584 of *Lecture Notes in Computer Science*, pages 24–28. Springer, 2003.

[15] Wai-Kau Lo and Vassos Hadzilacos. Using failure detectors to solve consensus in asynchronous shared-memory systems (extended abstract). In Gerard Tel and Paul M. B. Vitányi, editors, *WDAG*, volume 857 of *Lecture Notes in Computer Science*, pages 280–295. Springer, 1994.

[16] Michael E. Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000.

[17] Piotr Zielinski. Anti-Omega: the weakest failure detector for set agreement. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *PODC*, pages 55–64. ACM, 2008.